Abhrajyoti Kundu
Computer Science & IT (CS)

OPERATING SYSTEM-2 (GATE 2023) - REPORTS

OVERALL ANALYSIS | COMPARISON REPORT | SOLUTION REPORT

ALL(17) | CORRECT(9) | INCORRECT(7) | SKIPPED(1)

Q. 11

Have any Doubt ?

Consider the following pseudo code segment:

```
void function (const int i)
{
    int j = 1 – i;
    while (1)
    {
        while (turn ! = i);
        //critical section
        turn = j;
    }
}
```

'turn' is a global shared variable for above code. A process which contains above code get a value for i before call the 'function'. And value i can be 1 or 0 only. Consider following statements for above code segments:

S_1: Satisfy mutual exclusion requirement for only two processes with different values of i.

S_2: Satisfy bounded waiting requirement.

S_3: Satisfy progress requirement.

S_4: Satisfy mutual exclusion requirement for more than two processes.

Which of the above statements are correct?

A. S_1, S_2, S_3 — Your answer is **IN-CORRECT**

B. S_1 — Correct Option

Solution :

(b)

Let process P_0 get $i = 0$ and P_1 get $i = 1$.

And value of 'turn' can either 0 or 1. So at a time only one process can be in critical section among P_0 and P_1. And if more than two process, and let two process get $i = 0$, then when turn = 0 then there two processes will enter critical section.

So, S_1 is correct and S_2 is incorrect.

If only one process participating, and let its $i = 0$ and turn = 1, then this process never get into critical section and there is no one in critical section. So, it is not progressive. S_3 is incorrect.

Above code does not satisfy bounded waiting requirement, because if a process terminated in critical section and new process come with same i as terminated process, then it will go into critical section. Now, consider this new process also terminated in critical section. And this can go infinitely.

So, S_2 is incorrect.

C. S_2, S_3, S_4

D. S_3, S_1

QUESTION ANALYTICS +

Q. 12

Have any Doubt ?

Consider the following two pseudo-code segments:

Segment-I

```
wait (S3);
if (count == 0);
    wait (δ);
count ++;
signal (S3);
wait (S2);
{
    //section-1
}
signal (S2);
wait (α);
count --;
if (count == 0);
    signal (S1);
signal (β);
```

Segment-II

```
wait (S1);
{
    //section-2
}
signal (S1);
```

Semaphores S_1, S_2, S_3 and integer variable 'count' are shared data among processes who contains any one of the above code segment.

Assume, P is a set of processes which contain code segment-I and Q is a single process which contains code segment-II.

If Q is in 'section-2' then no process from set P can be in 'section-1'. And if a process of P is in 'section-1' then Q cannot be in 'section-2'. Maximum 3 processes of P can be simultaneously in

'section-1'. Initial value of S_1, S_2 and S_3 and count are x, y, 1 and 0. And we have to select some given semaphore in place of α, β, δ in segment-I.
Which of the following are correct values?

A
$x = 1$
$y = 3$
$\alpha = S_3$
$\beta = S_3$
$\delta = S_1$

Your answer is **Correct**

Solution :
(a)
(i) If Q is only ready to go in section-2, then value of S_1 should 1. So, $x = 1$ and it satisfy given conditions. And for this value if Q is in section-1 then $t \in P$ will have to wait for Q to finish, so δ should be S_1. $(\delta : S_1)$
(ii) If $t \in P$ come before Q then wait $(\delta : S_1)$ will stop Q to enter.
(iii) S_2 is only accessible on segment-1, and maximum three element of P are possible in section-1, so $S_2 = 3$ initially. So $y = 3$.
(iv) wait(S_3) and signal(S_3) force 'count++;' to execute atomically and it is necessary because if more than one element of P execute 'count++' simultaneously then it lead to race condition. So, we have to do same thing for 'count--;'. To do this $\alpha : S_3$ and $\beta : S_3$ are correct.
It is similar to reader-writer problem.

B
$x = 1$
$y = 1$
$\alpha = S_3$
$\beta = S_2$
$\delta = S_1$

C
$x = 1$
$y = 1$
$\alpha = S_3$
$\beta = S_3$
$\delta = S_1$

D
$x = 0$
$y = 1$
$\alpha = S_3$
$\beta = S_2$
$\delta = S_3$

QUESTION ANALYTICS +

Q. 13

Have any Doubt ?

Consider a system using one-to-one model of thread in which Kernel is aware of all threads and processes. Which of the following is incorrect for above system?

A Creation of a user-level thread in above system requires creation of corresponding Kernellevel thread.

Your answer is **IN-CORRECT**

B One-to-one model provides more concurrency on multiprocessor or multicore system than many-to-one thread model in which Kernel is not aware of user-level threads.

C Maximum number of user-level threads in a process possible in above system depends on the maximum Kernel-level threads possible in the system.

D If a user-level thread makes a blocking system call and other user-level threads are in ready state, then entire process of this thread will block in this system.

Correct Option

Solution :
(d)
(a) It is one-to-one model, so every user-level thread (ULT) need a Kernel-level thread (KLT).
(b) In one-to-one model Kernel aware of all ULTs, so it will allocate the other threads to other available processors/cores.
(c) Every system has limitation on maximum Kernel threads. And LLTs depend on KLT in one-to-one model.
(d) Here Kernel aware of ULTs, so if one thread blocks then Kernel consider it as thread call not process call and Kernel will block thread not process, and allocate other thread if any to the processor.

QUESTION ANALYTICS +

Q. 14

Have any Doubt ?

Consider the set of processes with arrival time (in ms), CPU burst time (in ms) shown below:

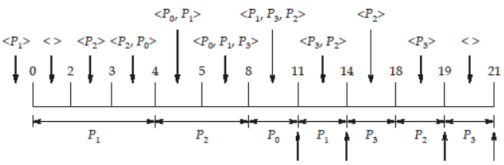
Process	Arrival time	Burst time
P_0	3	3
P_1	0	7
P_2	2	5

Above process executed on a CPU with Robin-Round scheduling with time quantum 4 ms. Average turn around time for above execution is _____ ms.

13.75 — Your answer is Correct13.75

Solution :
13.75

Use queue to solve above question. In below diagram < > represent queue and first element will delete first.



Turn around time (TA) = Finish time – Arrival time

$TA(P_0) = 11 - 3 = 8$ ms

$TA(P_1) = 14 - 0 = 14$ ms

$TA(P_2) = 19 - 2 = 17$ ms

$TA(P_3) = 21 - 5 = 16$ ms

$$\text{Average turn around} = \frac{8+14+17+16}{4} = \frac{55}{4} = 13.75 \text{ ms}$$

QUESTION ANALYTICS +

Q. 15 Have any Doubt ?

Consider the following C program is executed on a unix/linux system:

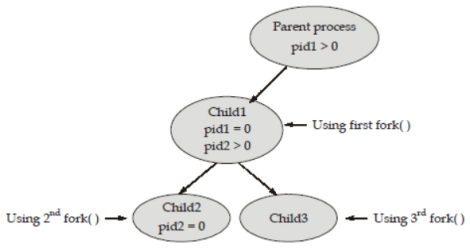
```
#include <stdio.h>
#include <unistd.h>
int main( )
{
    pid_t pid1 = fork( );
    if pid1 = = 0)
    {
        pid_t pid2 = fork( );
        if (pid2 > 0)
        {
            fork( );
        }
    }
    return 0;
}
```

Assume function call to fork( ) function never fail on above function program execution. How many total processes will create for above program?

4 — Correct Option

Solution :
4

Initially a single process for given program, which is parent process. Following diagram is showing execution pattern of program.



Above diagram has four processes parent, child1, child2, child3.
For statement 'pid1 = fork( )', parent process will get process id of child and child will get 0.

Your Answer is 8

QUESTION ANALYTICS +

Q. 16 Have any Doubt ?

Which of the following statement is/are correct?

Standard test-and-set() atomic instruction does not satisfy the bounded waiting requirement for — Correct Option

A Standard test_and_set( ) atomic instruction does not satisfy the bounded-waiting requirement for solution of critical_section problem. Correct Option

B A starvation situation possible in test_and_set( ) and compare_and_swap( ) based solution to critical-section problem. Correct Option

C Mutex lock is a hardware tool for solution to critical-section problem.

D Non-preemptive Kernel is free from race-conditions on Kernel data structures. Your option is **Correct**

YOUR ANSWER - d | CORRECT ANSWER - a,b,d | STATUS - ✗

Solution :

(a, b, d)
Both, test_and_set( ) and compare_and_swap( ) do not satisfy bounded-waiting requirement, because there is no condition on specific process. As, a result starvation possible in both solutions. So, (a), (b) are correct. Mutex lock is a software based solution. In non-preemptive Kernel, only one process can active in Kernel and other process cannot preempt this process. So, this process will come out of Kernel, after finish its work. So, no race condition, only sequential execution of processes.

QUESTION ANALYTICS +

Q. 17 Have any Doubt ?

Consider the set of processes with arrival time (in ms), CPU burst time (in ms), and priority (0 is highest priority) shown below:

Process	Arrival time	Burst time	Priority
P_0	1	1	0
P_1	0	3	1
P_2	3	3	3
P_3	5	6	2
P_4	7	5	4

None of the above processes have I/O burst time. Which of the following statements is/are correct for above given processes?

A If preemptive priority scheduling used, then only two processes preempted and they move from running state to ready state. Your option is **Correct**

B If non-preemptive priority scheduling used, then maximum processes in ready queue at an instance are two. Your option is **Correct**

C If preemptive priority scheduling used, then P_3 will terminate/finish before P_2. Your option is **Correct**

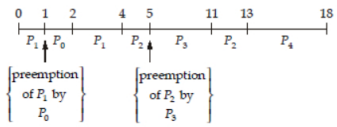
D If non-preemptive priority scheduling used, then P_4 will wait in ready queue for 6 ms. Your option is **Correct**

YOUR ANSWER - a,b,c,d | CORRECT ANSWER - a,b,c,d | STATUS - ✓

Solution :

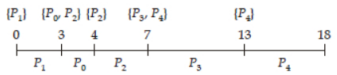
(a, b, c, d)

1. Preemptive priority scheduling



So, option (a) is correct. Here, P_3 finishes before P_2, so option (c) is correct.

2. Non-preemptive priority scheduling.



Here, two times, at 3 ms and 7 ms ready queue size is two. So, option (b) is correct.
Here, P_4 arrives at 7 ms and get CPU at 13 ms, so waiting time is 6 ms. So, option (d) is correct.

QUESTION ANALYTICS +